arm64e

An ABI for Pointer Authentication

LLVM Developers' Meeting
October 22nd, 2019

John McCall
Ahmed Bougacha

What is arm64e?

- arm64e is an ABI for pointer authentication on ARMv8.3
- ARMv8.3 is an AArch64 extension provided by the Apple A12 and later (e.g. iPhone XR/XS, released September 2018)
- Used for all system software on those devices
- Not ABI stable yet — still looking for ways to strengthen it

What is Pointer Authentication?

- Security mitigation technique
- Provides control flow integrity (CFI), limited data integrity
- Basic idea: sign and authenticate pointers to prevent attackers from escalating memory corruption bugs

Memory Corruption

- Many exploits start with memory corruption bugs
- e.g. buffer overflows, use-after-free
- Ideally, these bugs wouldn’t exist
  - Safe languages, safe practices, static analysis, thorough code review
- Practically, mitigation still has an important place

Exploitation

- Limited memory corruption is not usually the goal of an attack
- Attacker wants to access sensitive information, make specific system calls, exfiltrate data over network, etc.
- Escalating an attack often requires corrupting control flow

Code Payloads

- Attacker wants to run some custom code

- Can’t just write new instructions in modern systems

```
MOV   X0, #0x8             ; first argument: client socket descriptor
MOV   X1,  #0x1F0174ED0    ; second argument: address of password file in memory
MOV   X2, #8096            ; third argument: length
BL    _write
```

Gadgets

- Instead, attacker finds gadgets: bits and pieces of existing functions that collectively do what the attacker wants

```
MOV   X0, #0x8               ; first argument: client socket descriptor
MOV   X1,  #0x1F0174ED0      ; second argument: address of password file in memory
MOV   X2, #8096              ; third argument: length
BL    _write
```

Gadgets

- Instead, attacker finds gadgets: bits and pieces of existing functions that collectively do what the attacker wants

```
_getBitsInByte:
MOV   X0, #0x8             ; return number of bits in a byte
RET

_readPasswordHeader:
MOV   X17, #0x1F0174ED0    ; put address of password file in scratch register
LDR   X0, X17              ; load from it (leaving address in register)
RET

; next we need a gadget that will move x17 into x1
; etc.
```

ROP/JOP

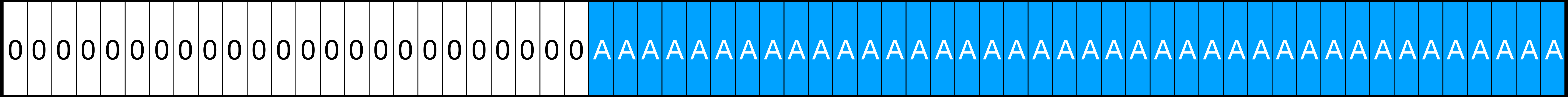
- Attacker must call all of these gadgets in the right sequence
- Use memory corruption to redirect indirect branches to gadgets
    - Redirecting returns: return-oriented programming (ROP)
    - Redirecting calls: jump-oriented programming (JOP)

Pointer Authentication

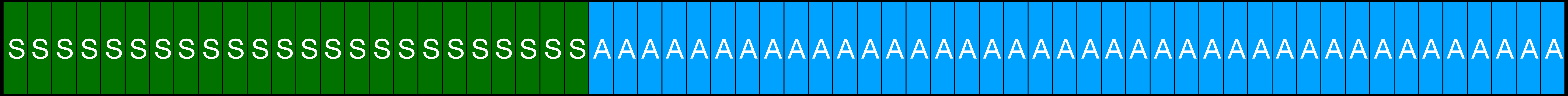
- Goal: prevent this from working by breaking attempts to redirect
- Add a signature to every code pointer
    - (and some select data pointers)
- Always authenticate signature before doing an indirect branch
    - (and some select loads)

ARMv8.3 Pointer Signatures

- Signature is stored in unused high bits of a 64-bit pointer (~25 bits today)



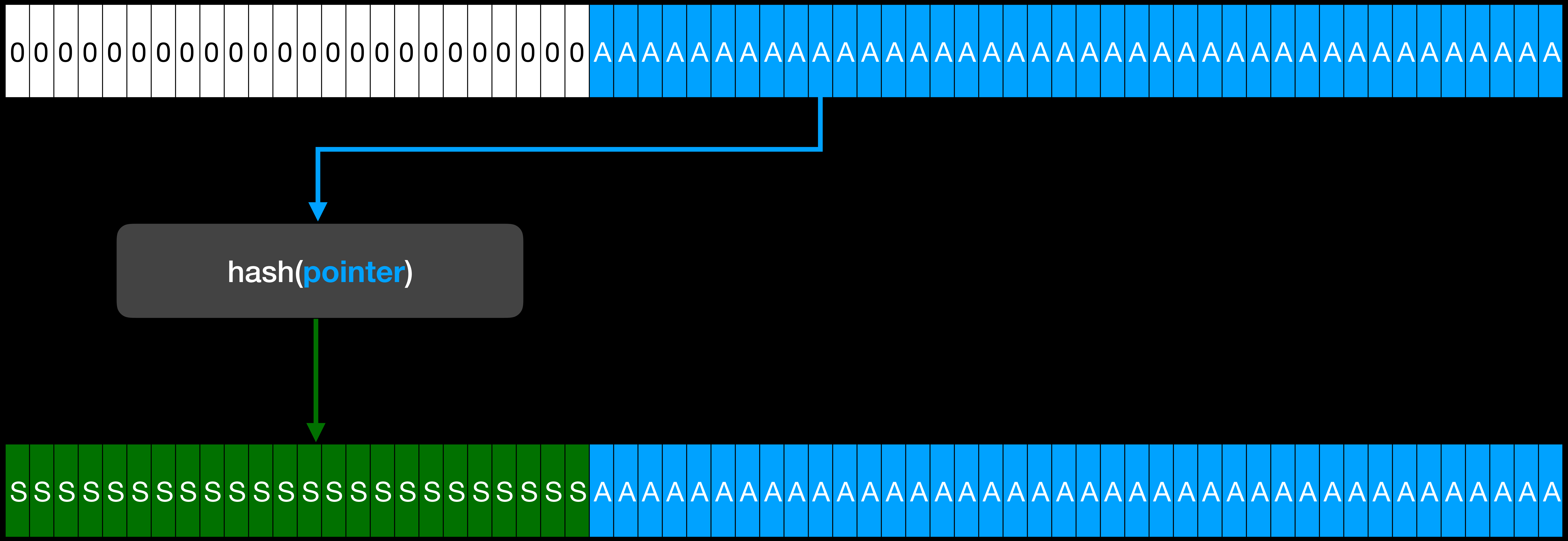
000000000000000000000000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA



SSSSSSSSSSSSSSSSSSSSSSSSAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

ARMv8.3 Pointer Signatures

- Computed by performing a cryptographic hash of the base pointer



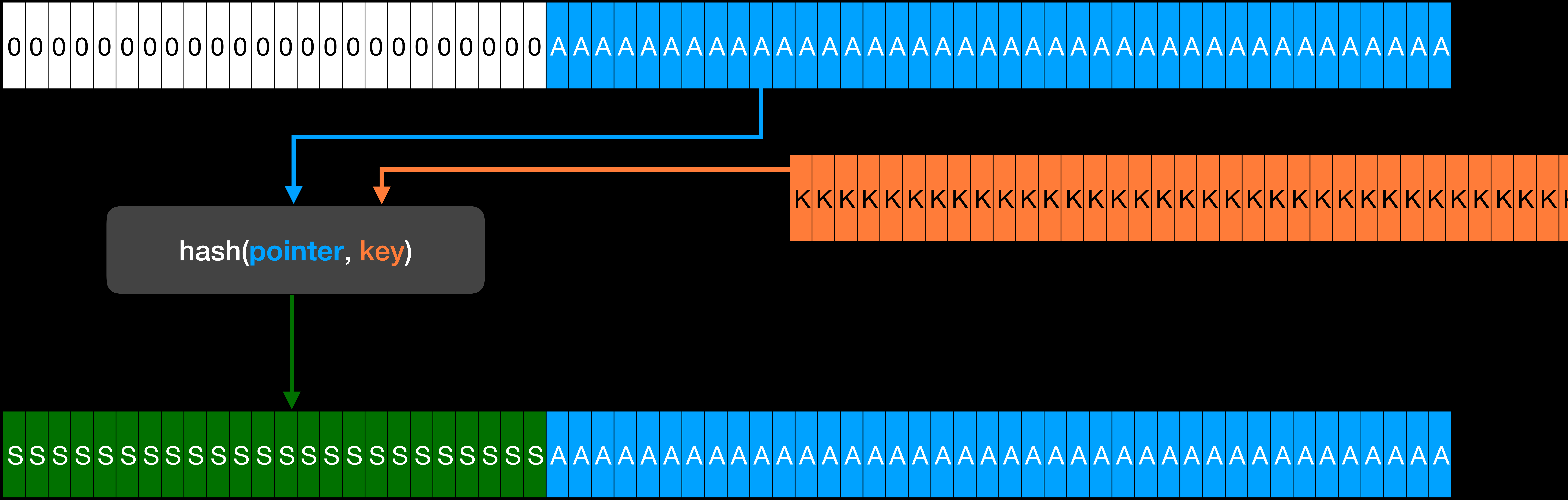
000000000000000000000000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

hash(pointer)

SSSSSSSSSSSSSSSSSSSSSSSSAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

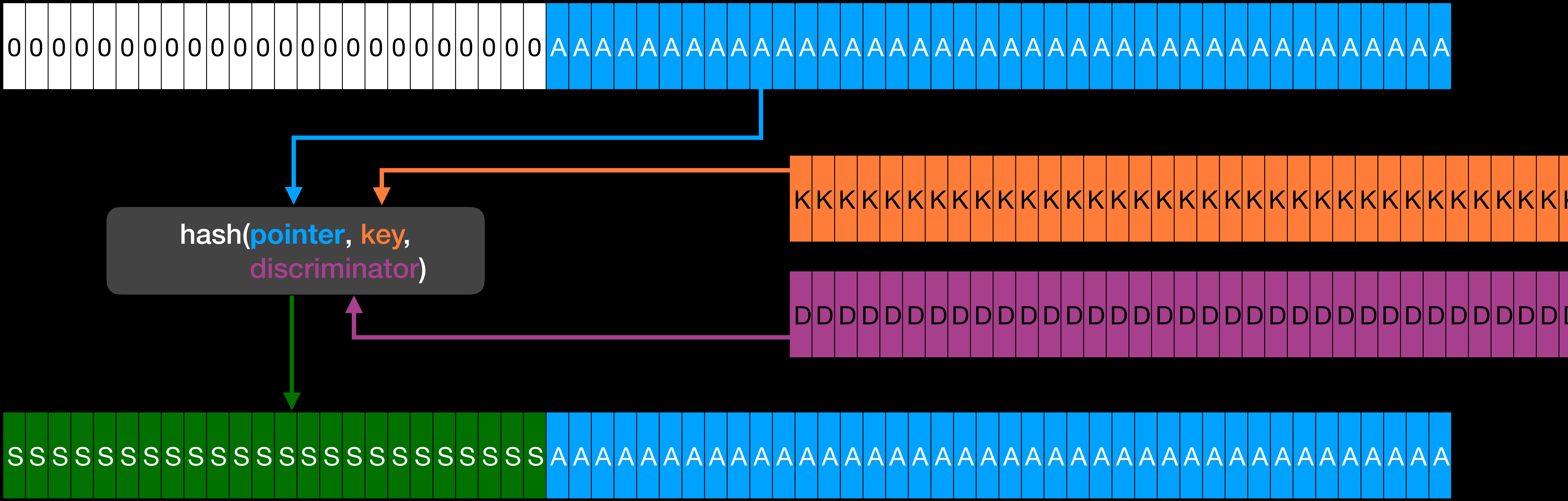
ARMv8.3 Pointer Signatures

- Hash also incorporates data from one of several secret 128-bit key registers, only directly readable by the kernel (a “pepper”)


000000000000000000000000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
KKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKK
hash(pointer, key)
SSSSSSSSSSSSSSSSSSSSSSSSAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

ARMv8.3 Pointer Signatures

- Hash also incorporates a 64-bit discriminator (a “salt”)



000000000000000000000000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
hash(pointer, key,
discriminator)
KKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKK
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
SSSSSSSSSSSSSSSSSSSSSSSSAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Pointer Substitution

- Signing with secret key means attackers can’t forge signed pointers

- Attackers can still overwrite signed pointers with other signed pointers

- Means gadgets have to be whole functions, but apparently that’s not a serious hurdle

Discriminators

- Substitution only works if all the inputs to the hash are the same

```
hash(pointer, key, discriminator)
```

- Small number of keys, so it mostly comes down to discriminators

Discriminators

- Ideally, every different “purpose” would use a different discriminator
    - A pointer should only authenticate if a human programmer would say that the pointer was meant to be used there
- Pointer authentication mostly driven automatically by compiler
    - Limited by imperfect knowledge
    - Limited by language design

Language ABI

- Compiler automatically protects all indirect branches:
  - `return`
  - `switch`
  - symbol imports (GOT)
  - C function pointers
  - C++ virtual functions
  - etc.
- ABI rule specifies key and how to compute the discriminator

Discriminators in the ABI

- ARMv8.3 allows discriminators to be arbitrary 64-bit values
- For practical reasons, discriminators used in language ABI are restricted
- Combination of two factors:
  - whether to use address diversity
  - choice of small constant discriminator

Address Diversity

- Discriminator includes storage address of pointer

- Same pointer stored in different places will have a different signature

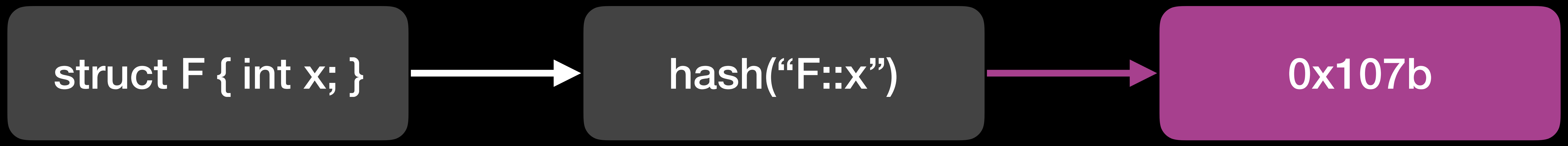
- Copying requires re-signing, so attackers can’t replace pointers themselves, have to convince the program to do it for them

- Incompatible with `memcpy`, makes copies much more expensive

Constant Discriminators

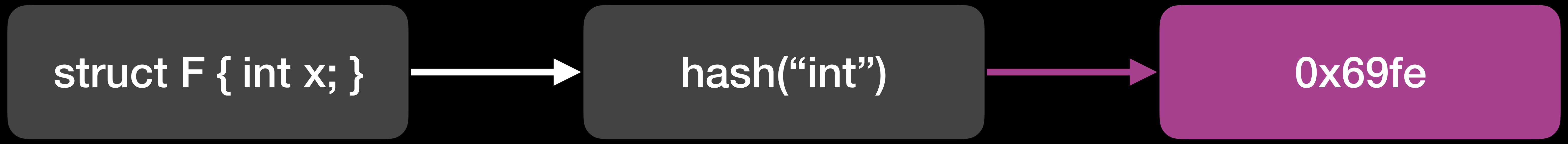
- 16-bit constant integer

- Can be derived from declaration:



struct F { int x; } → hash("F::x") → 0x107b

- Can be derived from type:



struct F { int x; } → hash("int") → 0x69fe

- Declaration is better, but can't break abstract, type-based uses

Example: C++ Virtual Functions

- No direct access to v-table in language, ODR provides strong guarantees
- Can sign virtual function pointers with address diversity
- Can use mangling of method declaration for constant discriminator
- Abstract uses (member function pointers) can be supported without weakening basic ABI
- V-table pointer in object also signed

Example: C Function Pointers

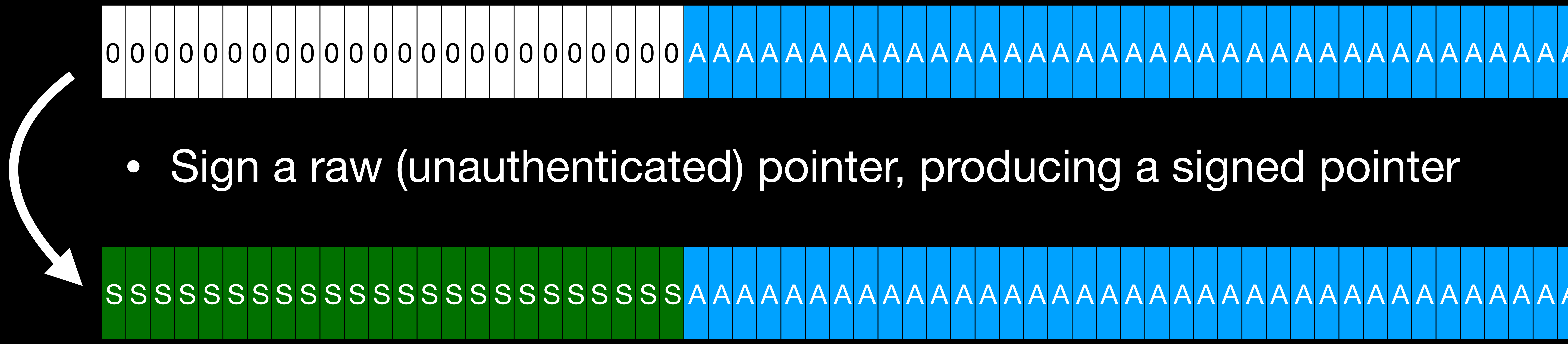
- Pointers must be copyable with `memcpy`, so no address diversity
- Can take address of function-pointer variables, so must use common discriminator for function-pointer type
- Lots of practical deployment challenges with discriminating by type
- Currently using a common discriminator of 0 for all C function pointers
- Clang provides language features to opt in to better discrimination

Generating Code for arm64e

Core Operations

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A

Core Operations

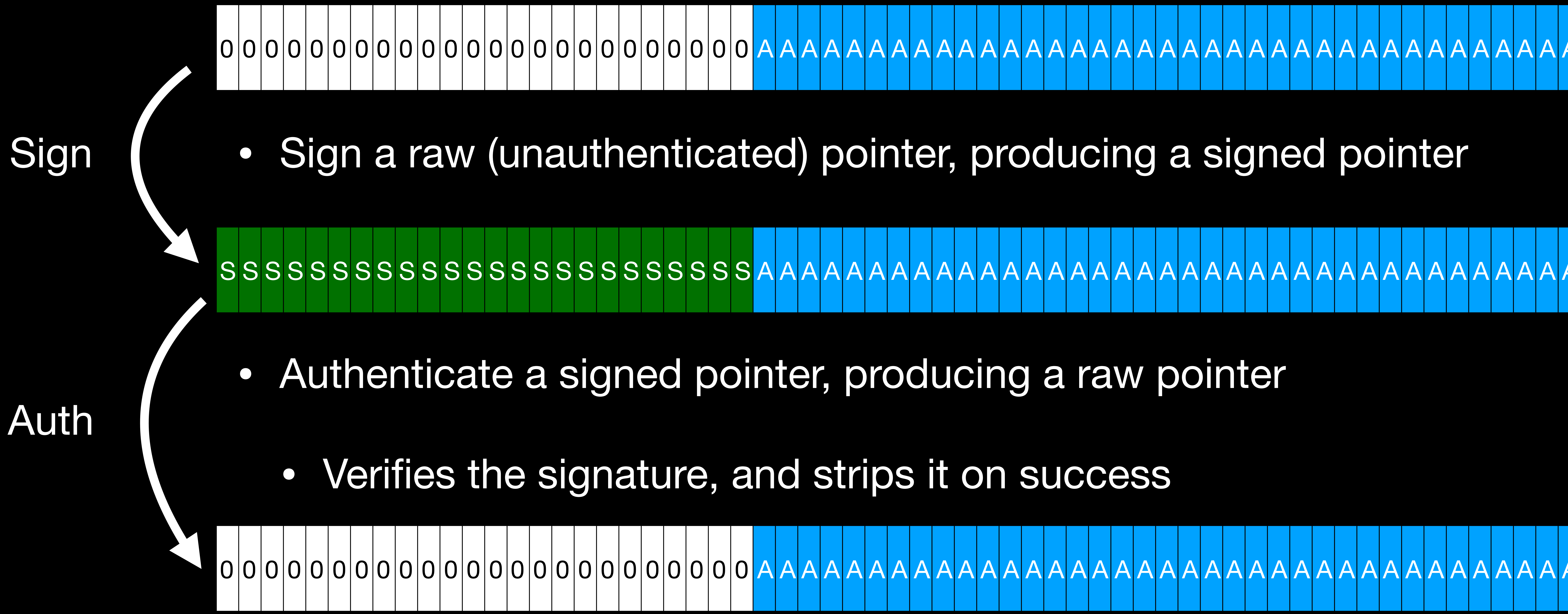


000000000000000000000000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

- Sign a raw (unauthenticated) pointer, producing a signed pointer

SSSSSSSSSSSSSSSSSSSSSSSSAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Core Operations



000000000000000000000000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Sign

- Sign a raw (unauthenticated) pointer, producing a signed pointer

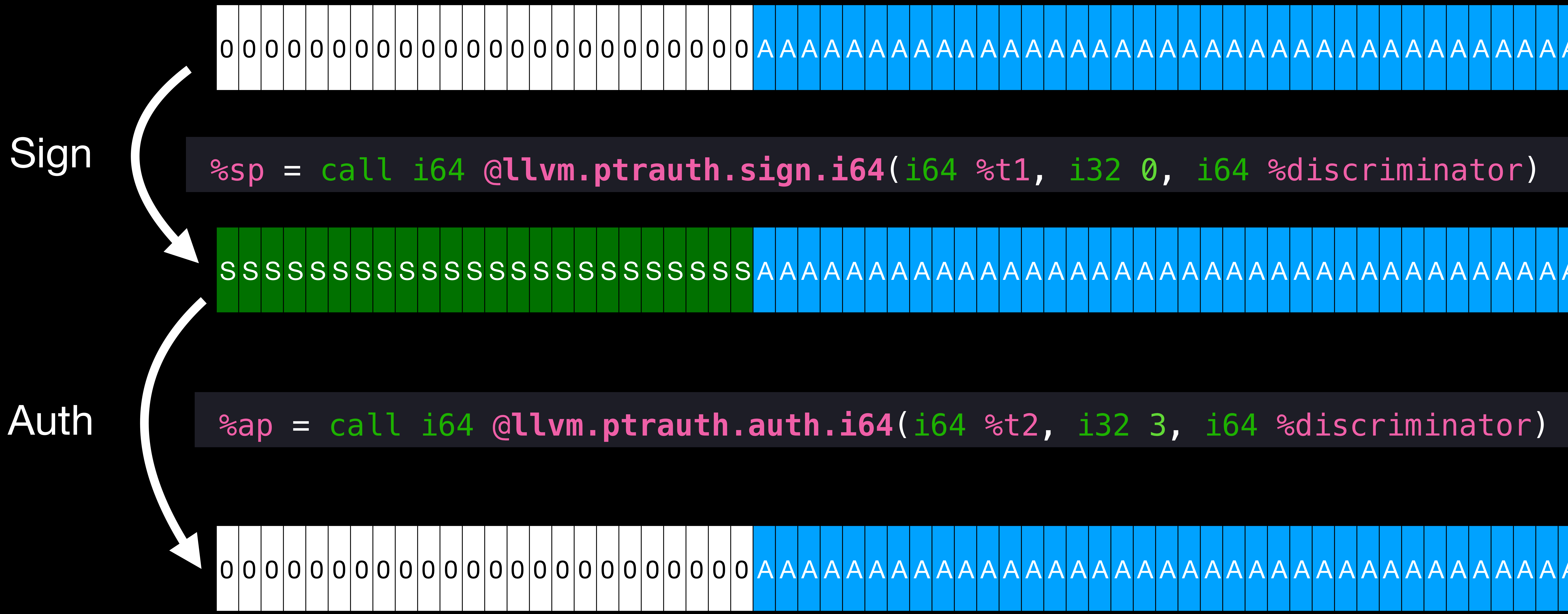
SSSSSSSSSSSSSSSSSSSSSSSSAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Auth

- Authenticate a signed pointer, producing a raw pointer
  - Verifies the signature, and strips it on success

000000000000000000000000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

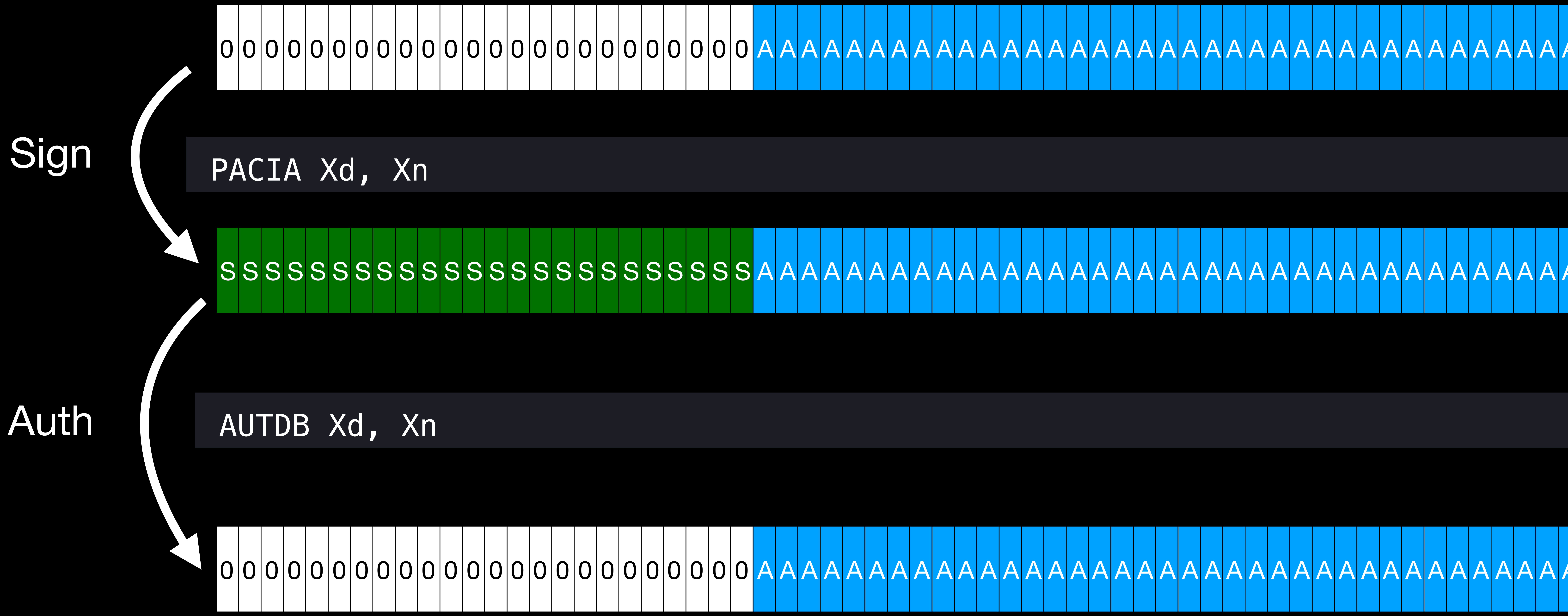
Core Operations


000000000000000000000000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Sign
SSSSSSSSSSSSSSSSSSSSSSSSAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Auth
000000000000000000000000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

```
%sp = call i64 @llvm.ptrauth.sign.i64(i64 %t1, i32 0, i64 %discriminator)
```

```
%ap = call i64 @llvm.ptrauth.auth.i64(i64 %t2, i32 3, i64 %discriminator)
```

Core Operations



000000000000000000000000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Sign

```
PACIA Xd, Xn
```

SSSSSSSSSSSSSSSSSSSSSSSSAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

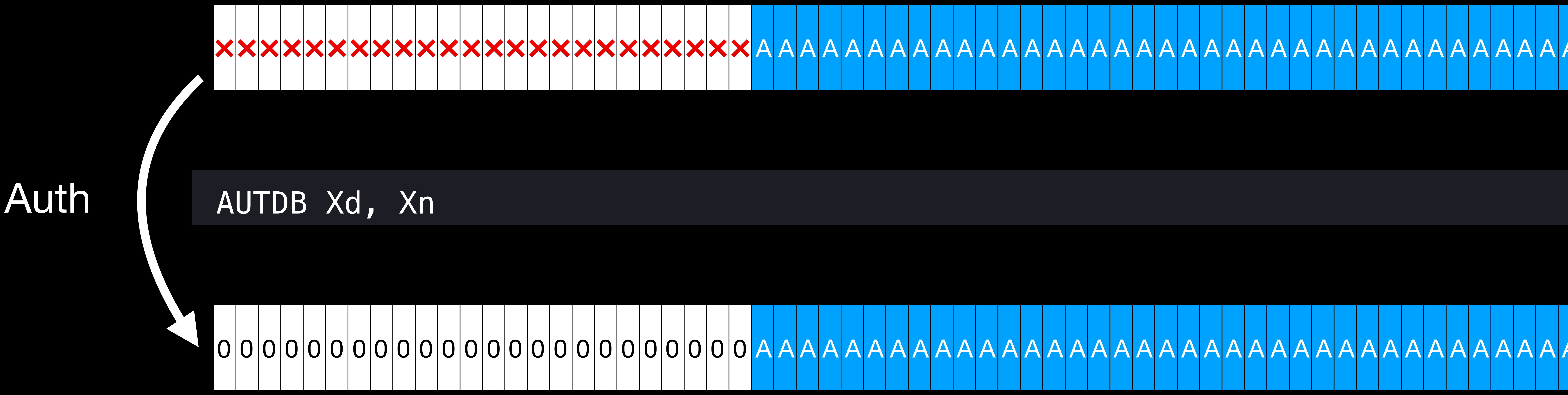
Auth

```
AUTDB Xd, Xn
```

000000000000000000000000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Core Operations

- Pointers with an invalid signature can't be authenticated



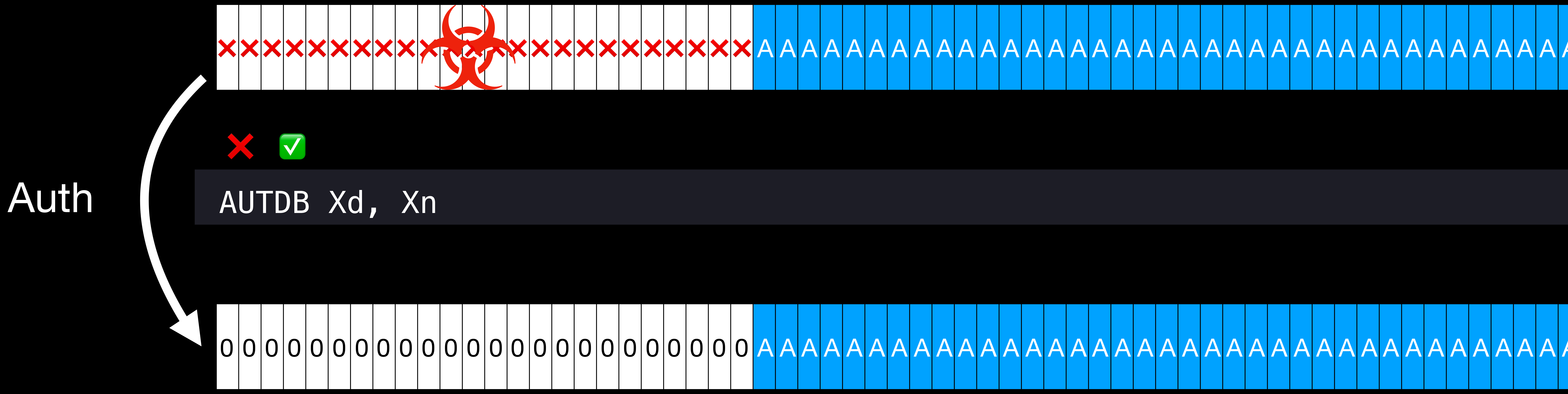
××××××××××××××××××××××××AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

```
AUTDB Xd, Xn
```

000000000000000000000000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Core Operations

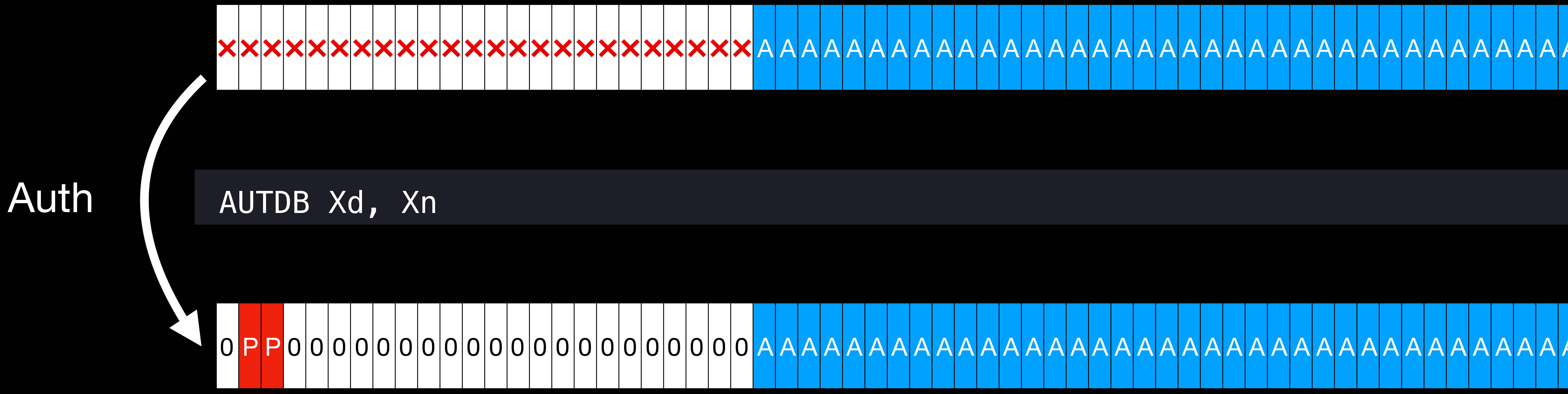
- Pointers with an invalid signature can't be authenticated


××××××××××××××××××××××××AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Auth
✗ ✅
000000000000000000000000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

```
AUTDB Xd, Xn
```

Core Operations

- Pointers with an invalid signature can't be authenticated


××××××××××××××××××××××××AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Auth
0PP000000000000000000000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

```
AUTDB Xd, Xn
```

Security Requirements

- Auth: must prevent attackers from bypassing signature verification
- Sign: must prevent attackers from signing pointers they control
- Core operations deal with raw pointers
  - Raw pointers are vulnerable, because they aren't verified
  - Raw pointers shouldn't be exposed (spilled to memory, ...)

Security Guarantees

- It's hard to reason about arbitrary uses
    - No guarantees can be made (e.g., against spilling)
- But we can reason about certain critical, well-defined, uses
    - arm64e mainly uses auth/sign to implement Control Flow Integrity
    - We must guarantee integrity of pointers used in control flow

Important Use-cases

- Authenticate a pointer...
  - ...used as a branch/call target
  - ...that's immediately re-signed
- Sign a pointer...
  - ...to a constant, as a constant initializer
  - ...to a constant, in code

Important Use-cases

- Authenticate a pointer...
  - ...used as a branch/call target
  - ...that's immediately re-signed
- Sign a pointer...
  - ...to a constant, as a constant initializer
  - ...to a constant, in code

```
(*funptr)();

obj->method();
```

Auth Operand Bundle: Call

- `"ptrauth"` operand bundle on indirect `calls`

```
call void %signed_callee() [ "ptrauth"(i32 0, i64 %disc) ]
```

- Guarantees integrity of the intermediate pointer

- On ARMv8.3, guarantees combined instruction codegen:

```
BLRAAZ Xd
```

Auth Operand Bundle: IndBr

- `indirectbr` is also indirect control flow
  - Let's give it a "`ptrauth`" operand bundle
- Tedious but straightforward patch

Auth Operand Bundle: Switch?

- Jump tables are created late
  - Jump table dispatch only exists in the backend
- We could sign the jump table entries
  - ...would require moving them from text to data
  - ...would prevent shrinking them for small offsets
- Too expensive

Jump Table Hardening

- Jump-table dispatch sequences are hardened using a custom sequence:

```
CMP   Xindex, #<jt size>
CSEL  Xindex, Xindex, XZR, ls         ; range-check the index
; we don't control the index: it could have been spilled across arbitrary blocks
; on index overflow, it's okay to pick any case: it's legitimate control flow
ADRP  Xjt, _JT0@PAGE
ADD   Xjt, _JT0@PAGEOFF               ; materialize the jump table address
LDRSW Xoffset, [Xjt, Xindex, lsl #2]  ; load the offset from the table
ADD   Xtarget, Xjt, Xoffset           ; compute the target
BR    Xtarget                         ; jump to it: no auth, because it's safe
```

Important Use-cases

- Authenticate a pointer...
  - ...used as a branch/call target
  - ...that's immediately re-signed
- Sign a pointer...
  - ...to a constant, as a constant initializer
  - ...to a constant, in code

```
void (*p)(char *);

return ((void)(*)(int *)) p;
```

Resign

- Authenticate a pointer using key/discriminator A, and re-sign it using key/discriminator B

```
declare i64 @llvm.ptrauth.resign.i64(i64, i32, i64, i32, i64)
```

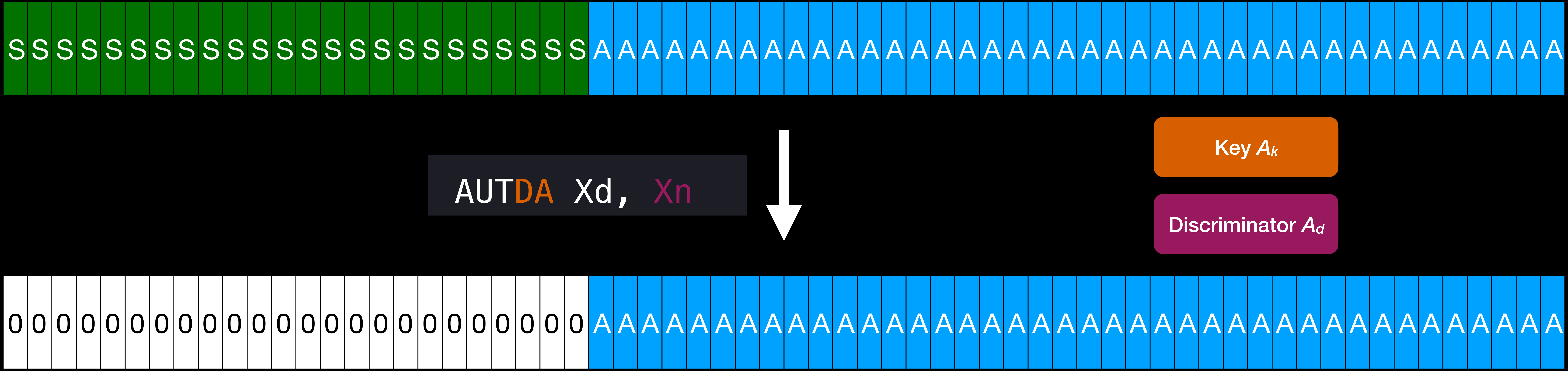
- Guarantees integrity of the intermediate pointer:

```
AUTDA X16, Xn
PACDB X16, Xm
```

Resign

S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A

Resign



SSSSSSSSSSSSSSSSSSSSSSSSAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

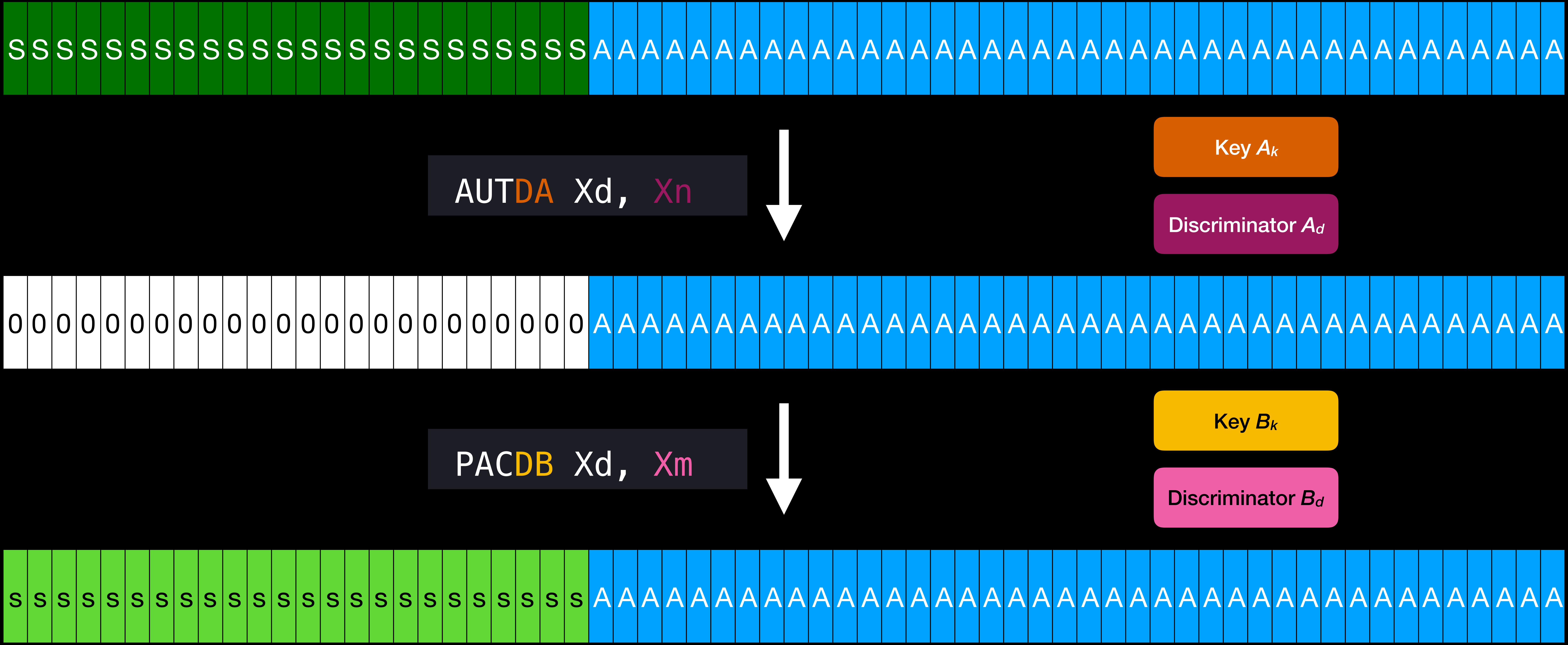
```
AUTDA Xd, Xn
```

Key A_k

Discriminator A_d

000000000000000000000000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Resign



SSSSSSSSSSSSSSSSSSSSSSSSAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

```
AUTDA Xd, Xn
```

Key A_k

Discriminator A_d

000000000000000000000000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

```
PACDB Xd, Xm
```

Key B_k

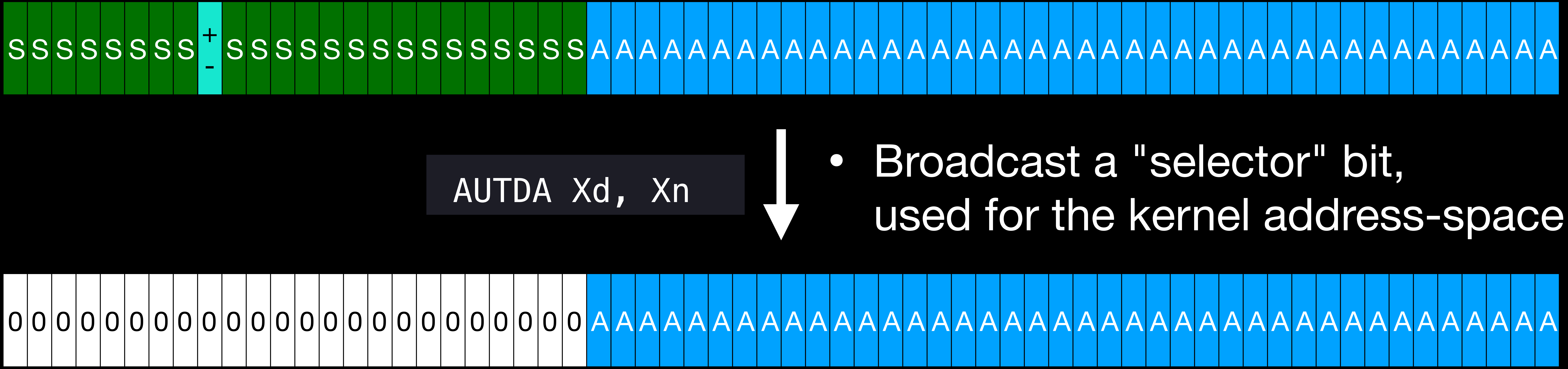
Discriminator B_d

SSSSSSSSSSSSSSSSSSSSSSSSAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Resign

S	S	S	S	S	S	S	S	+ -	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A

Resign



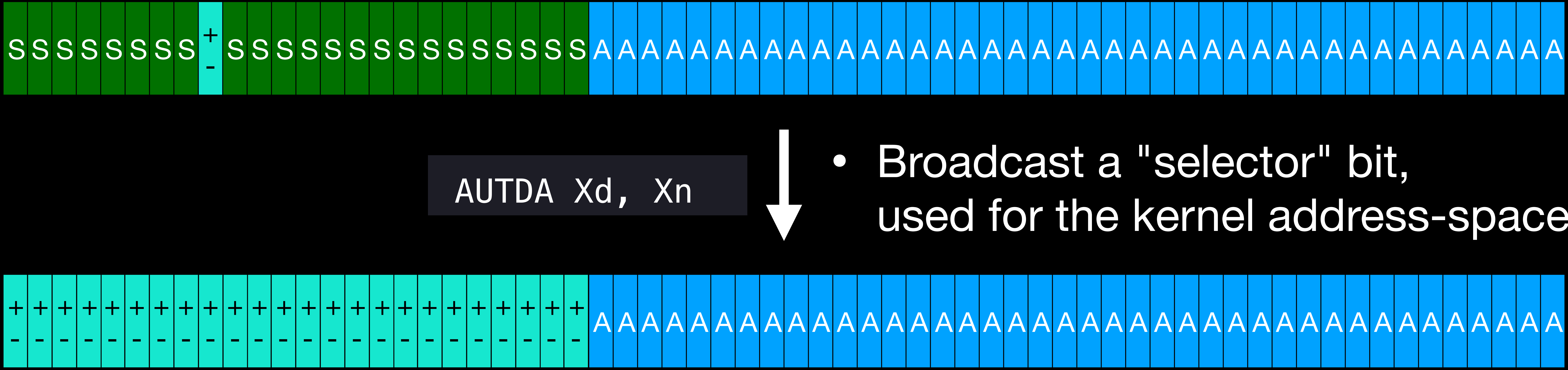
S	S	S	S	S	S	S	S	+ -	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A

```
AUTDA Xd, Xn
```

- Broadcast a "selector" bit, used for the kernel address-space

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A

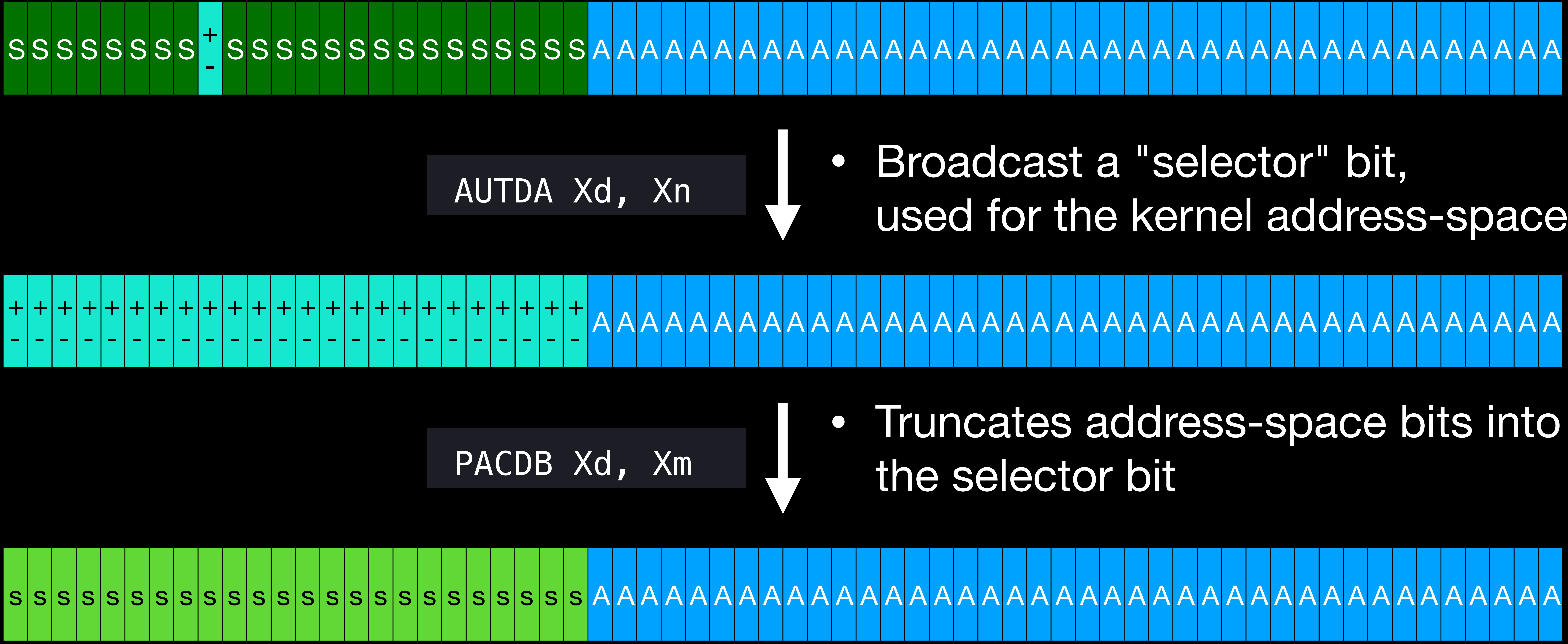
Resign


S S S S S S S S ± S S S S S S S S S S S S S S S A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
± A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A

```
AUTDA Xd, Xn
```

- Broadcast a "selector" bit, used for the kernel address-space

Resign



SSSSSSSS+-SSSSSSSSSSSSSSSAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
+-AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
SSSSSSSSSSSSSSSSSSSSSSSSAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

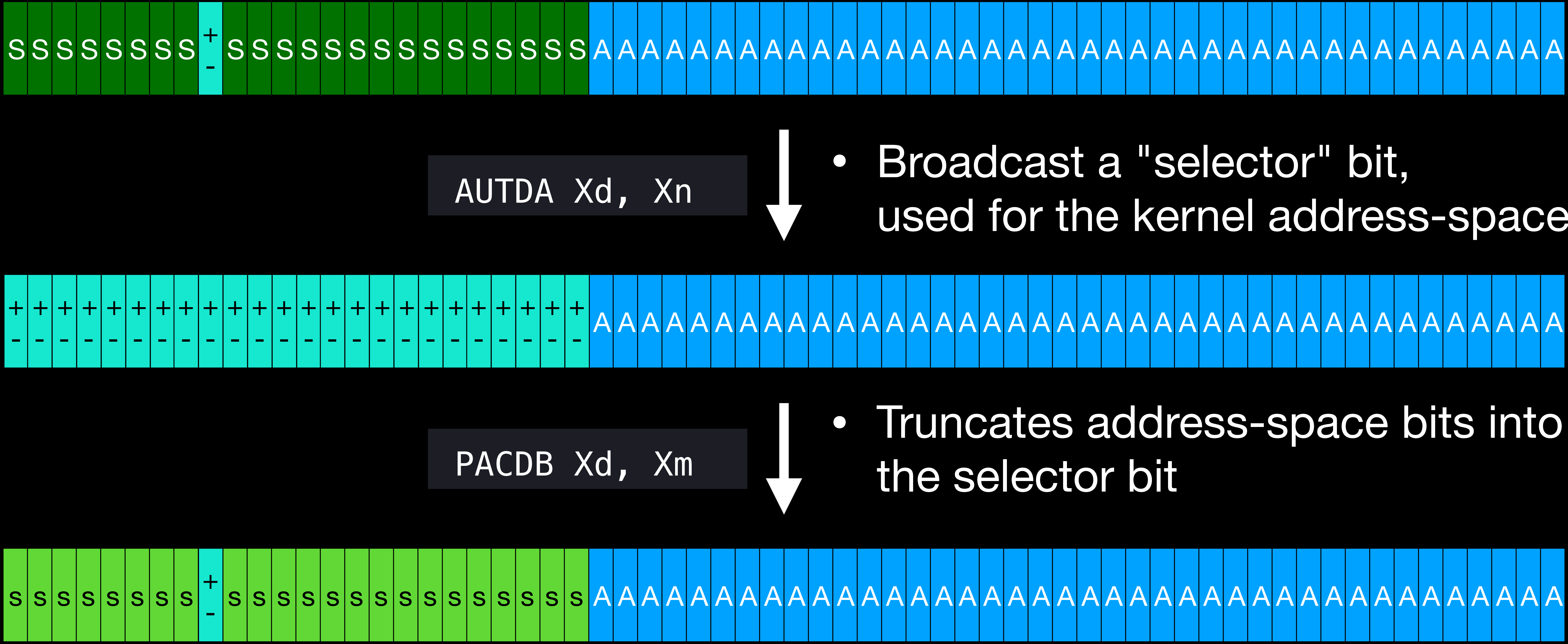
```
AUTDA Xd, Xn
```

- Broadcast a "selector" bit, used for the kernel address-space

```
PACDB Xd, Xm
```

- Truncates address-space bits into the selector bit

Resign



SSSSSSSS
+
-
SSSSSSSSSSSSSSS
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
++++++++++++++++++++++++

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ssssssss
+
-
sssssssssssssss
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

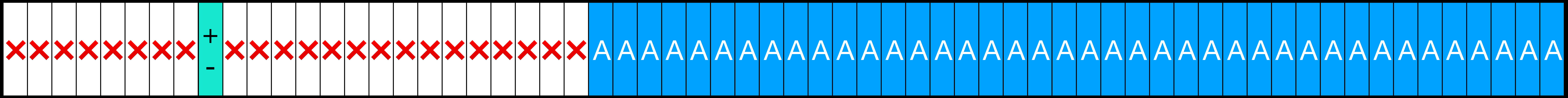
```
AUTDA Xd, Xn
```

- Broadcast a "selector" bit, used for the kernel address-space

```
PACDB Xd, Xm
```

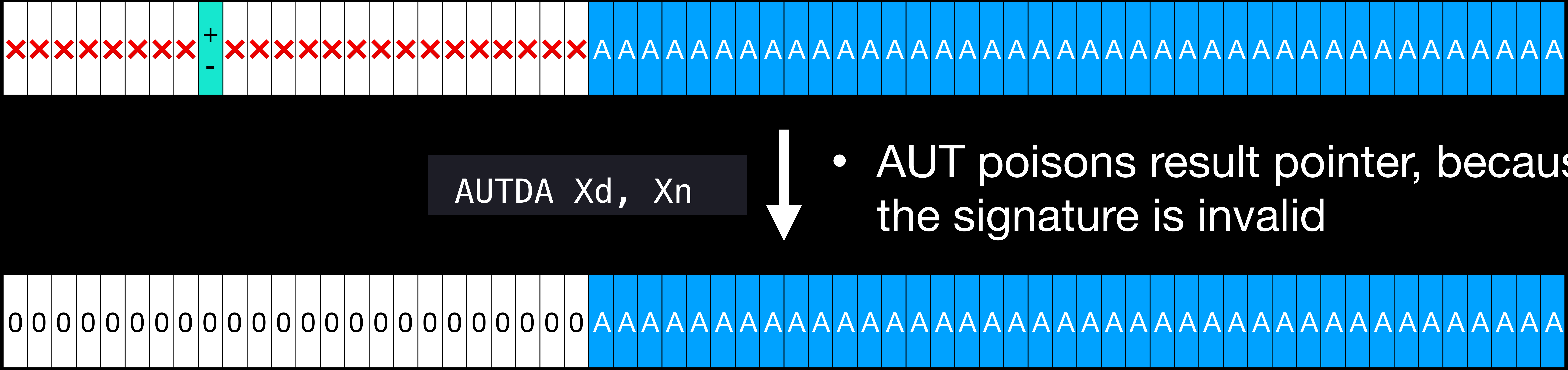
- Truncates address-space bits into the selector bit

Resign Failures



✗	✗	✗	✗	✗	✗	✗	✗	+ -	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A

Resign Failures



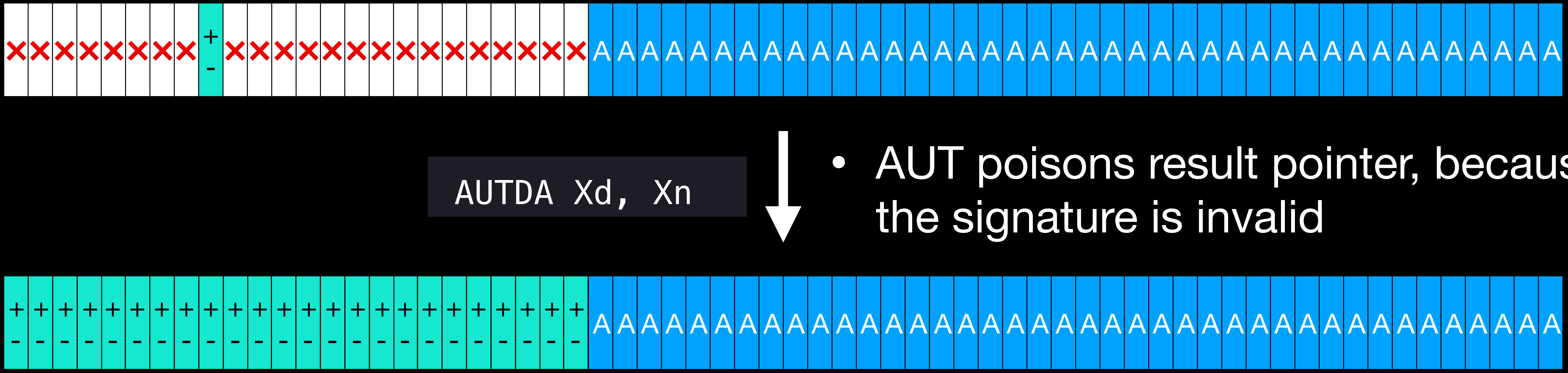
××××××××+-×××××××××××××××AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

```
AUTDA Xd, Xn
```

000000000000000000000000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

- AUT poisons result pointer, because the signature is invalid

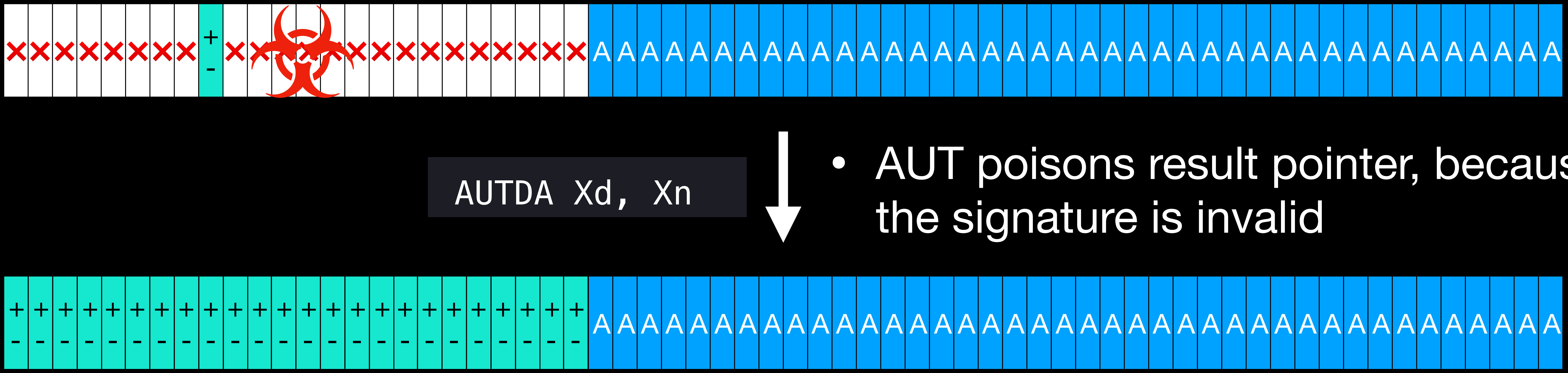
Resign Failures



```
AUTDA Xd, Xn
```

- AUT poisons result pointer, because the signature is invalid

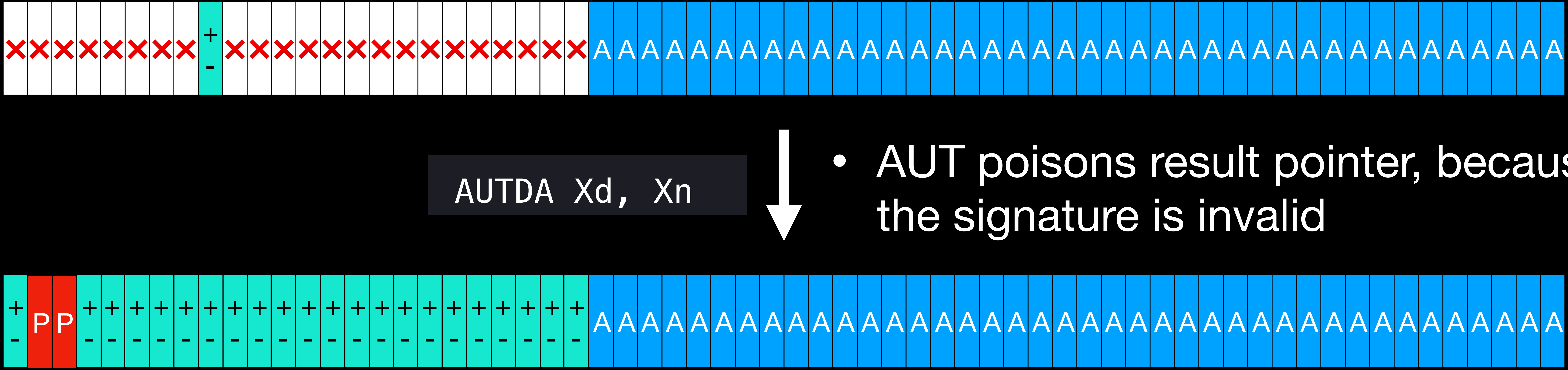
Resign Failures


××××××××+-×××××××××××××××AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
+-AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

```
AUTDA Xd, Xn
```

- AUT poisons result pointer, because the signature is invalid

Resign Failures

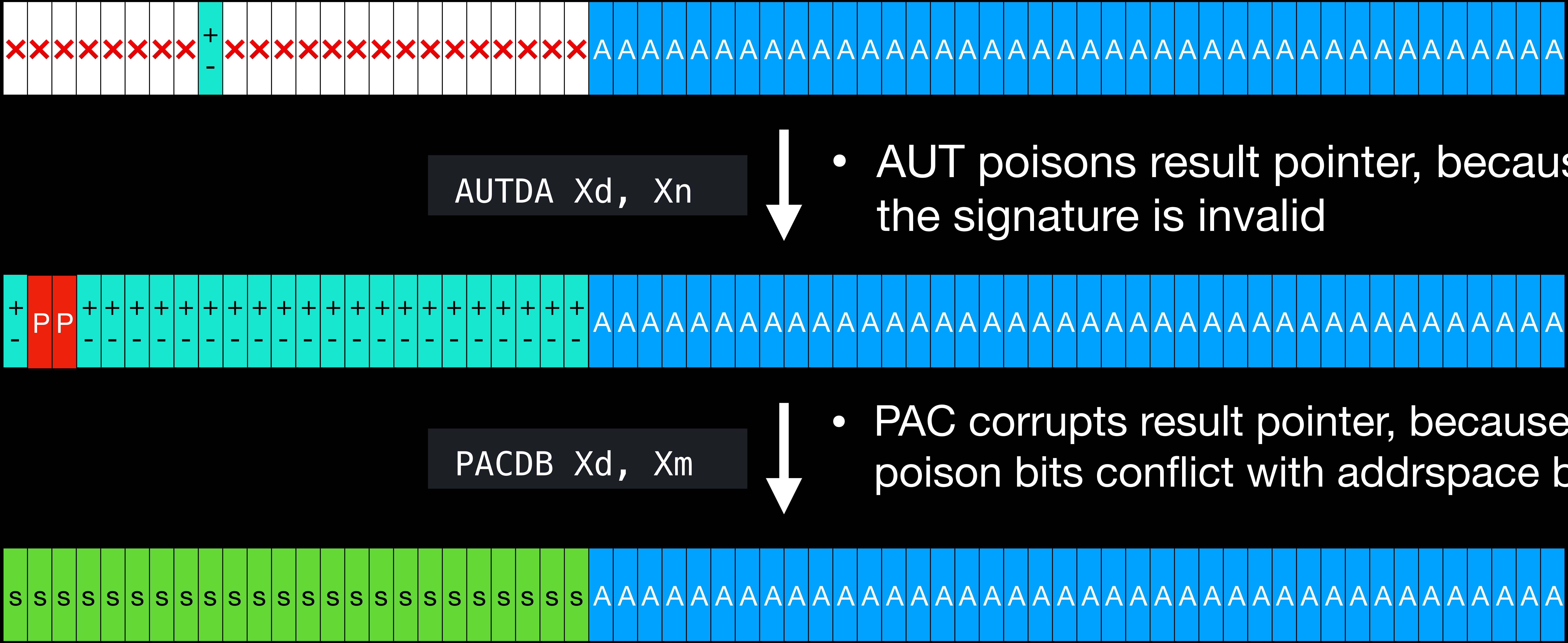

××××××××
+
-
×××××××××××××××
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
+
-
PP
+++++++++++++++++++++

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

```
AUTDA Xd, Xn
```

- AUT poisons result pointer, because the signature is invalid

Resign Failures



× × × × × × × × +- × × × × × × × × × × × × × × × A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
+- P P +- A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
S S S S S S S S S S S S S S S S S S S S S S S S A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A

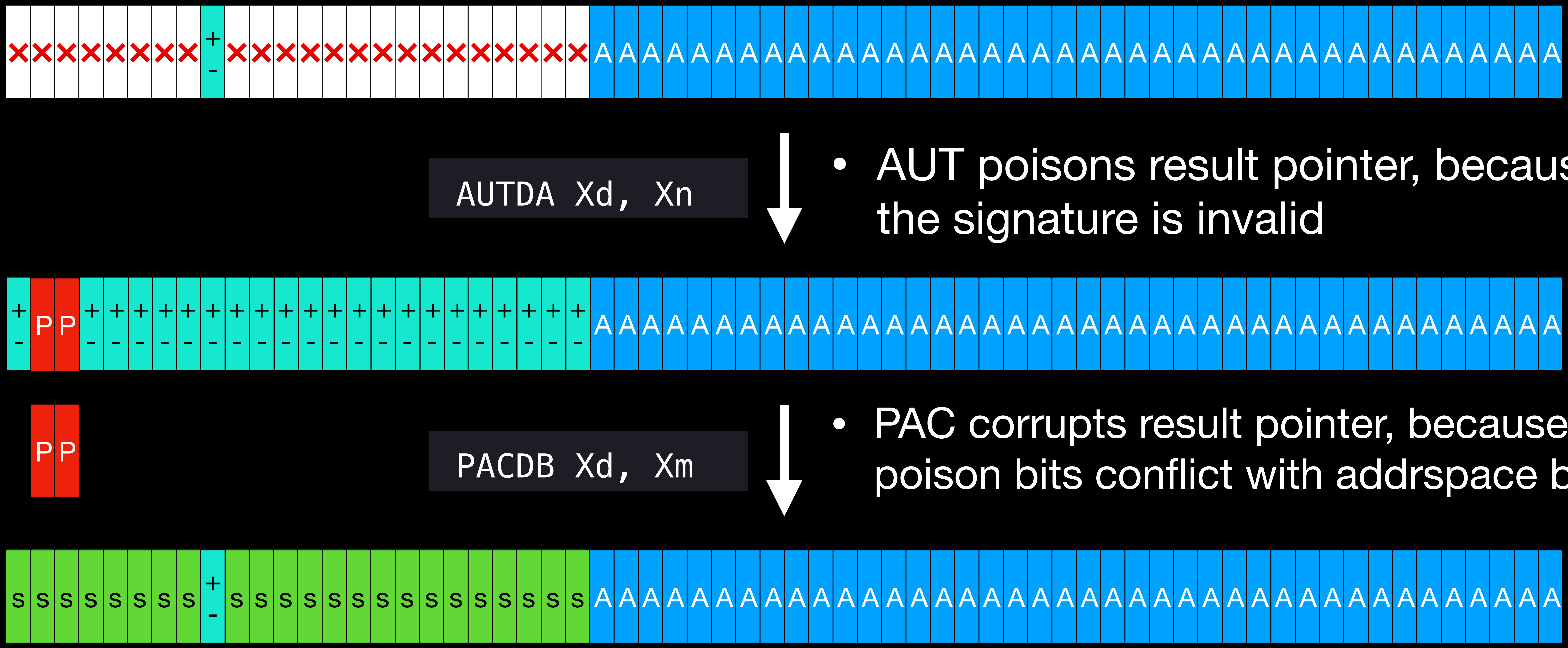
```
AUTDA Xd, Xn
```

- AUT poisons result pointer, because the signature is invalid

```
PACDB Xd, Xm
```

- PAC corrupts result pointer, because poison bits conflict with addrspace bits

Resign Failures


× × × × × × × × +- × × × × × × × × × × × × × × × A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
+- P P +- A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
P P
S S S S S S S S +- S S S S S S S S S S S S S S S A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A

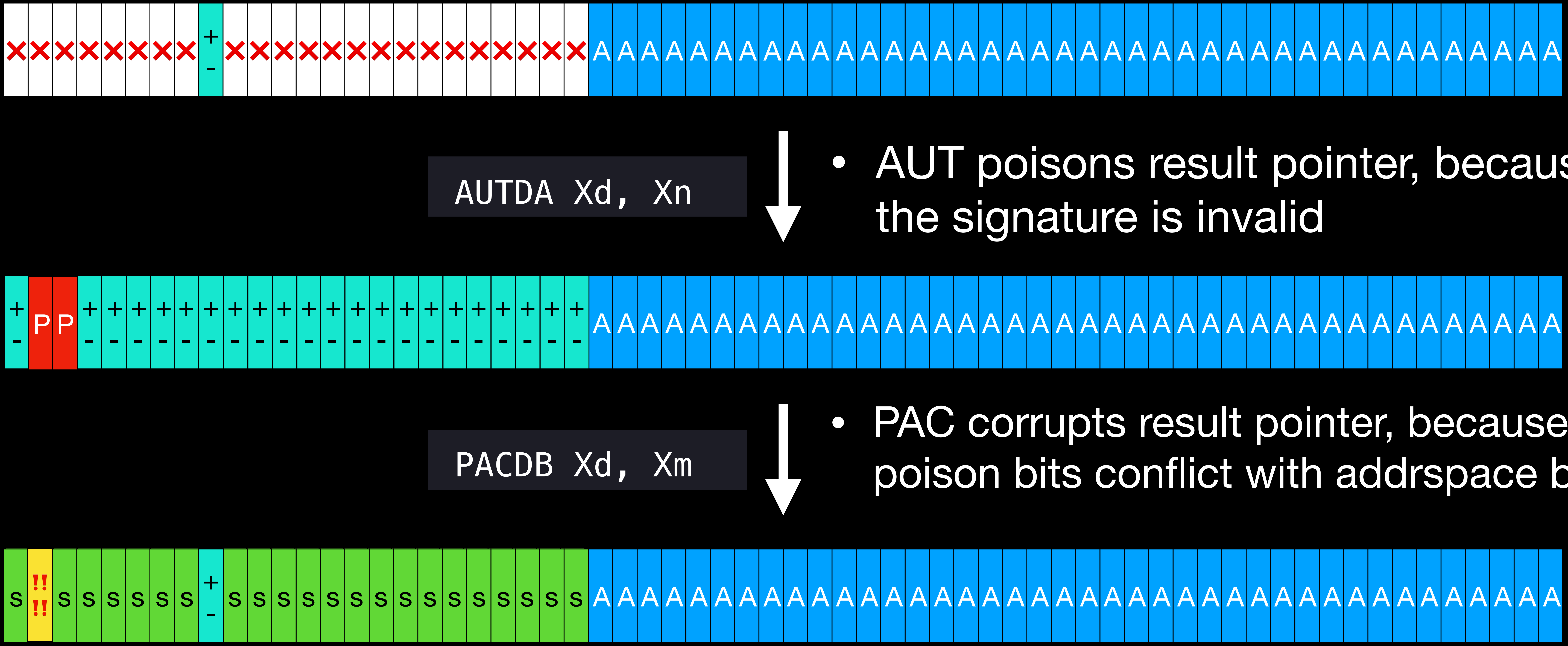
```
AUTDA Xd, Xn
```

- AUT poisons result pointer, because the signature is invalid

```
PACDB Xd, Xm
```

- PAC corrupts result pointer, because poison bits conflict with addrspace bits

Resign Failures



× × × × × × × × + - × × × × × × × × × × × × × × × A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
+ - P P + - A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
S !! !! S S S S S S + - S S S S S S S S S S S S S S S A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A

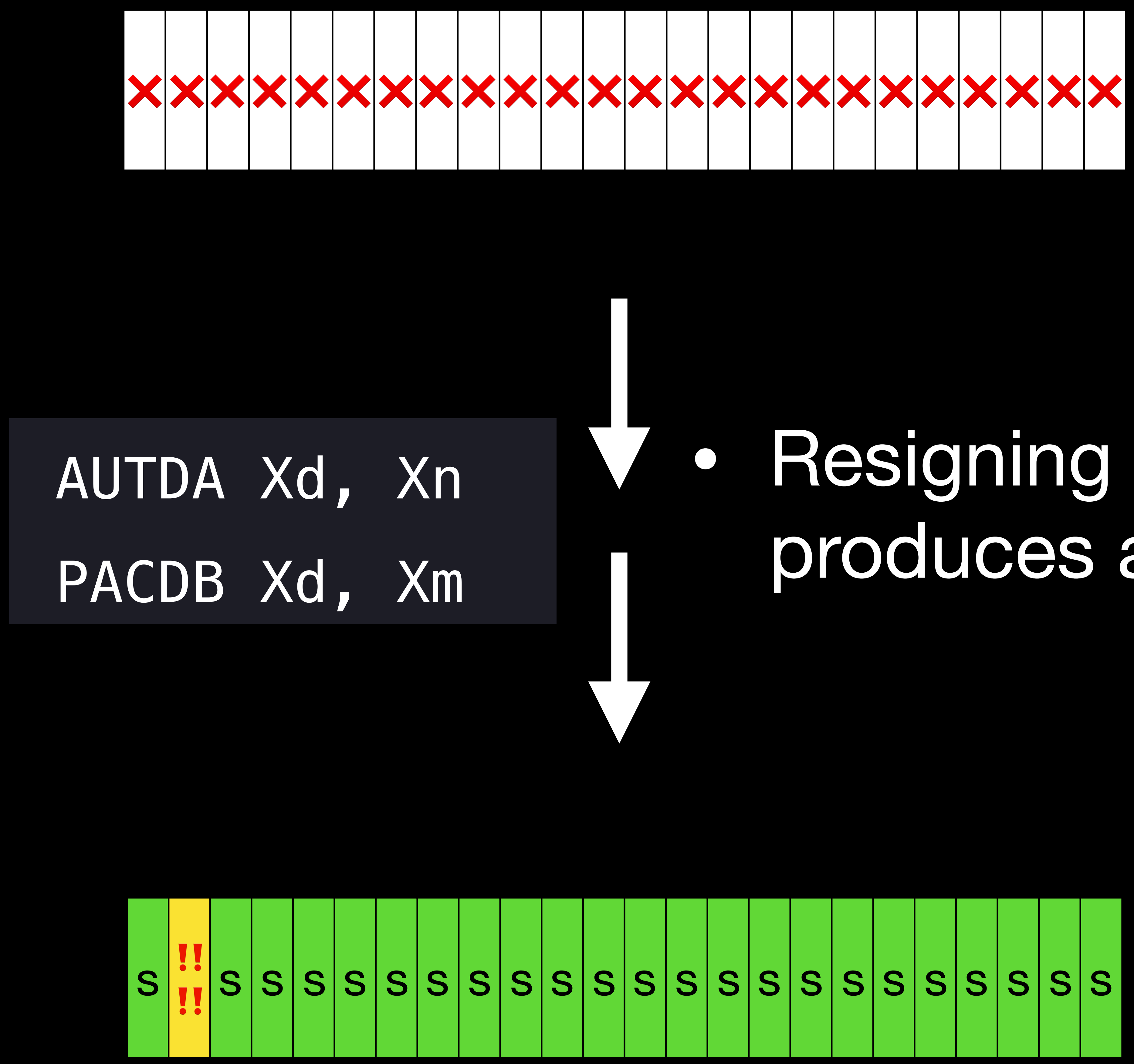
```
AUTDA Xd, Xn
```

- AUT poisons result pointer, because the signature is invalid

```
PACDB Xd, Xm
```

- PAC corrupts result pointer, because poison bits conflict with addrspace bits

Resign Problem #1



```
AUTDA Xd, Xn
PACDB Xd, Xm
```

- Resigning an invalidly-signed pointer produces a validly-signed pointer!

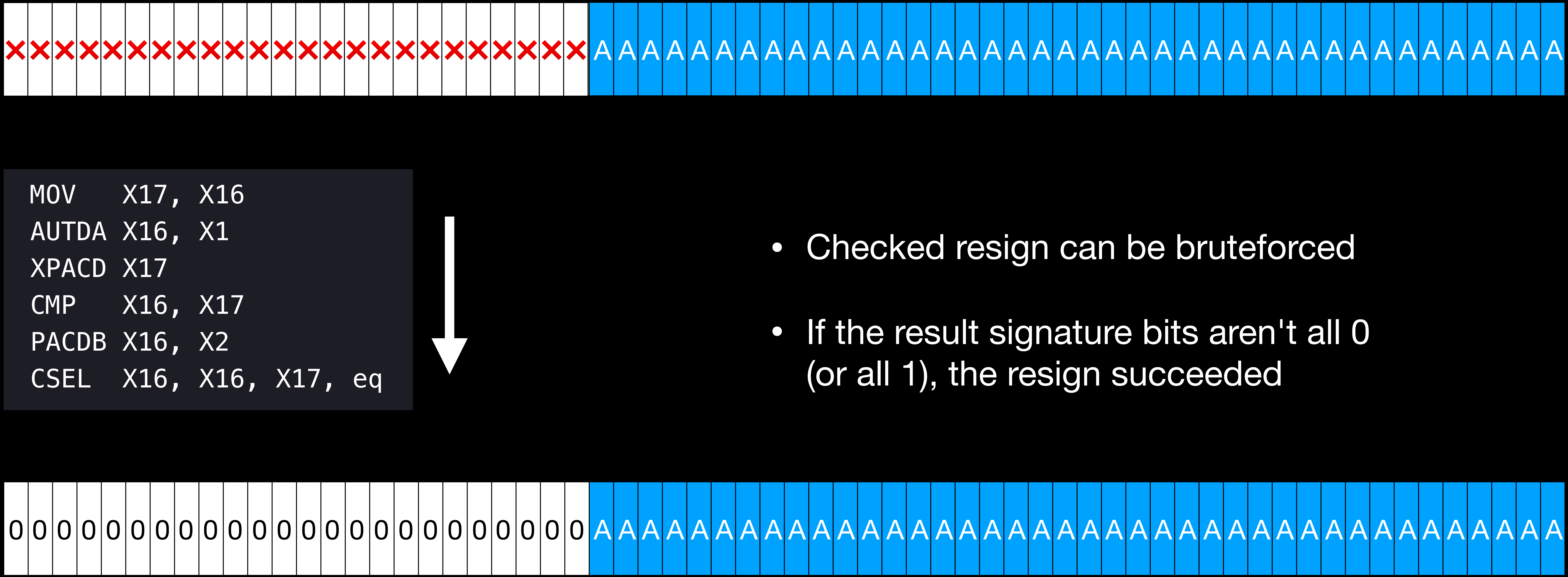
Resign Mitigation #1

- Resign sequence should check for AUT failure

- And return a pointer with no leaked signature bits

```
MOV   X17, X16             ; We'll need a copy of the pointer
AUTDA X16, X1              ; Authenticate it
XPACD X17                  ; But strip the signature from the copy
CMP   X16, X17             ; Compare the two
PACDB X16, X2              ; Sign the result
CSEL  X16, X16, X17, eq    ; On strip/auth mismatch: return the stripped value
```

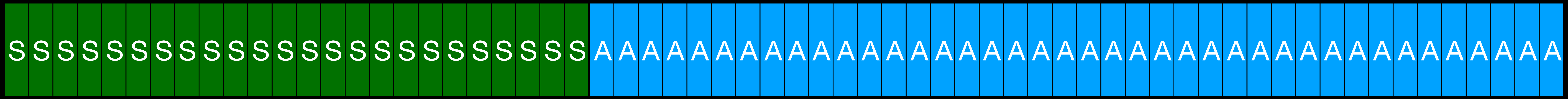
Resign Problem #2


××××××××××××××××××××××××AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
000000000000000000000000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

```
MOV   X17, X16
AUTDA X16, X1
XPACD X17
CMP   X16, X17
PACDB X16, X2
CSEL  X16, X16, X17, eq
```

- Checked resign can be bruteforced
- If the result signature bits aren't all 0 (or all 1), the resign succeeded

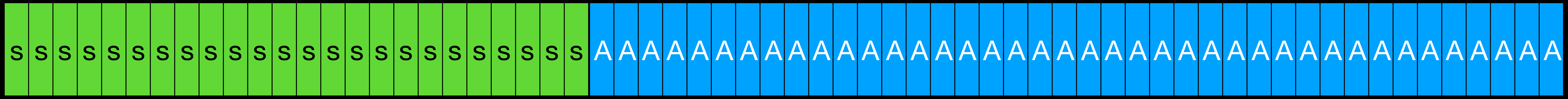
Resign Problem #2


SSSSSSSSSSSSSSSSSSSSSSSSAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

```
MOV   X17, X16
AUTDA X16, X1
XPACD X17
CMP   X16, X17
PACDB X16, X2
CSEL  X16, X16, X17, eq
```

↓ ✗ ✅

- Checked resign can be bruteforced
- If the result signature bits aren't all 0 (or all 1), the resign succeeded


ssssssssssssssssssssssssAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Resign Mitigation #2

- Resign sequence shouldn't be bruteforceable
    - Not a problem for most auths: the result is (really) used immediately
- Resign sequence should trap on auth failure

```
 MOV   X17, X16             ; We'll need a copy of the pointer
 AUTDA X16, X1              ; Authenticate it
 XPACD X17                  ; But strip the signature from the copy
 CMP   X16, X17             ; Compare the two
 B.EQ  Lsuccess             ; On success, move on
 BRK   #0xc472              ; On mismatch, trap!
Lsuccess:
 PACDB X16, X2              ; Sign the result
```

Important Use-cases

- Authenticate a pointer...
  - ...used as a branch/call target
  - ...that's immediately re-signed
- Sign a pointer...
  - ...to a constant, as a constant initializer
  - ...to a constant, in code

```
typedef void (*fnptr_t)(char *);

fnptr_t actions[] = { &f1, &f2 };
```

Signed Pointer Constant

- `llvm.ptrauth` Authenticated "wrapper" Global (ideally a ConstantExpr)

```
@f.ptrauth = private constant { i8*, i32, i64, i64 }
    { i8* bitcast (i8()* @f to i8*), i32 <key>, i64 <addr disc>, i64 <disc> },
    section "llvm.ptrauth"

@signed_f = constant i8()* bitcast ({ i8*, i32, i64, i64 }* @f.ptrauth to i8()*)
```

- Lowered to a new mach-o relocation:

```
_signed_f:
  .quad _f@AUTH(ia,1234,addr)
```

Important Use-cases

- Authenticate a pointer...
  - ...used as a branch/call target
  - ...that's immediately re-signed
- Sign a pointer...
  - ...to a constant, as a constant initializer
  - ...to a constant, in code

```
void f(char *);

return &f;
```

Signed Pointer Materialization

- `llvm.ptrauth` globals can be used in code too:

```
ret i8()* bitcast ({ i8*, i32, i64, i64 }* @f.ptrauth to i8()*)
```

- Which we lower to:

```
ADRP X16, _f@PAGE
ADD X16, X16, _f@PAGEOFF    ; materialize the pointer, the Darwin way
PACIA X16, Xn               ; sign it
```

Signed Pointer Materialization

```
ADRP X16, _f@PAGE
ADD X16, X16, _f@PAGEOFF      ; materialize the pointer
PACIA X16, Xn                 ; sign it
```

- Prevent transforms from exposing the intermediate pointer
  - Backend uses combined ops (PtrAuthGA in ISel, pseudo in AArch64)
- Prevent OS exceptions from exposing the intermediate register value
  - The compiler always uses x16/x17 for "sensitive registers"
  - The kernel guarantees the integrity of x16/x17 on exceptions

Important Use-cases

- Authenticate a pointer...
  - ...used as a branch/call target
  - ...that's immediately re-signed
- Sign a pointer...
  - ...to a constant, as a constant initializer
  - ...to a constant, in code

arm64e

- An ABI for Pointer Authentication
- Extends arm64 language ABIs to provide CFI
    - Discriminator choice is constrained, but is the key to hardening
- Exposes interesting compiler problems
    - Integrity must be preserved throughout all transformations
- Not ABI stable yet — still looking for ways to strengthen it

arm64e

An ABI for Pointer Authentication

LLVM Developers' Meeting
October 22nd, 2019

John McCall
Ahmed Bougacha